# Towards a UML/MARTE CSP-OZ Based Software Development Methodology

Mourad Maouche, Mohamed Bettaz

Philadelphia University

{*mmaouch, mbettaz*}*@philadelphia.edu.jo*

**IFIP WG1.3 Workshop, Eindhoven Meeting, Netherlands**

March 31, 2016

## Overview

## Introduction

- study potential synergies between distributed simulation (DS) and MDE technologies
- Topcu et al., Distributed Simulation: A MDE Approach (Simulation Foundations, Methods and Applications), Springer (2016), provide a comprehensive review of **DS** from the **MDE** perspective
- we focus on distributed multiscale modeling and simulation (DMMS)
- objectif: contribute to bridging the gap between
  - an "e-science" community (dealing with modeling and simulation of complex -natural world- phenomema) and a
  - SE community (using rigorous approaches to address the design and implementation of complex systems)

## The proposed Methodology

- MD
- based on the integration of UML/MARTE, CSP-OZ, PyCSP
  - UML/MARTE for the requirements
  - CSP-OZ for the design
  - PyCSP for the implementation
- use of institution theory to ensure soundness of the integration
- motivation
  - first motivate the use of UML/MARTE for the development of DMMS systems
  - second motivate the use of the combination (UML/MARTE, CSP-OZ, PyCSP)

## Motivating the Use of UML/MARTE

- investigations show four main reasons:
  1. both of (UML/MARTE and DMMS modelers) use similar modeling methodology (separation of concerns, following the so-called Y structure design methodology)
     - UML/MARTE: application modeling, HW/SW execution platform design
     - DMMS: modeling of e-science phenomena, design of simulation engines
  2. both of them advocate a component-based approach
  3. UML/MARTE supports concurrency, synchronization and communication features
     - useful for the modeling of distributed and concurrent simulation systems
  4. UML/MARTE defines an explicit model for the (logical/physical) time concept
     - might cover the development of both event-driven and time-driven simulators

## Motivating the Use of UML/MARTE, CSP-OZ and PyCSP

- sharing of core abstractions
- making the transformation between various formalisms straightforward

## Related Works

- several contributions advocate the use of appropriate (modeling and programming) languages for various views of the systems under design
  - for Bjorner et al., 40 Years of Formal Methods: Some Obstacles and Some Possibilities?, LNCS (2014), "the trend is to develop verification technology around programming languages" and that "verificating frameworks will be part of programming IDEs"
  - Hennicker et al., A Generic Framework for Multi-Disciplinary Environmental Modeling, proceedings of the iEMS (2010), propose an integration of UML, process algebra and Java
  - in the position paper Towards an Institutional Framework for Heterogeneous Formal Development in UML, LNCS 8950, Springer (2015), Knapp et al., consider the use of UML, OCL, ACSL and C
  - Michael Moller et al., Linking CSP-OZ with UML and Java, Springer (2004), use a combination of UML, CSP-OZ, JML, Java, and jassda

## More on DMMS

- foundations
  - formalization: scales, phenomena, domains, models, single scale models (sub-models), conduits, filters, mappers, observations, etc.
  - J.Borgdorff, E. Lorenz, C. Bona-Casas, J.-L. Falcone, B. Chopard (University of Geneva), A. G. Hoekstra (University of Amsterdam, National Research University ITMO, Saint-Petersburg)
- implementation of an environment (MUSCLE: MUltiScale Coupling Library and Environment)
  - simulation engines execute models independently of the progamming languages used to implement them (C, C++, Java, Fortran)
  - M. Mamonski, B. Bosak, K. Kurowski (Poznan Supercompting and Networking Center), M. Ben Belgacem (University of Geneva), D. Groen, P.V. Coveny (University College London)

## Overview of MUSCLE

- library: submodels are implemeted with some specific APIs
- configuration: submodels are instantiated and coupled
- runtime environment: submodels are executed on a variety of machines

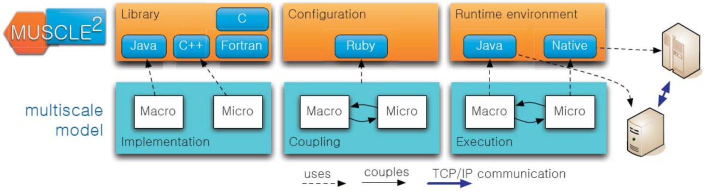


Figure : Source: Joris Borgdorff et al./Procedia Computer Science 18(2013)

## MUSCLE: Illustration of the Modeling Methodology (1/4)

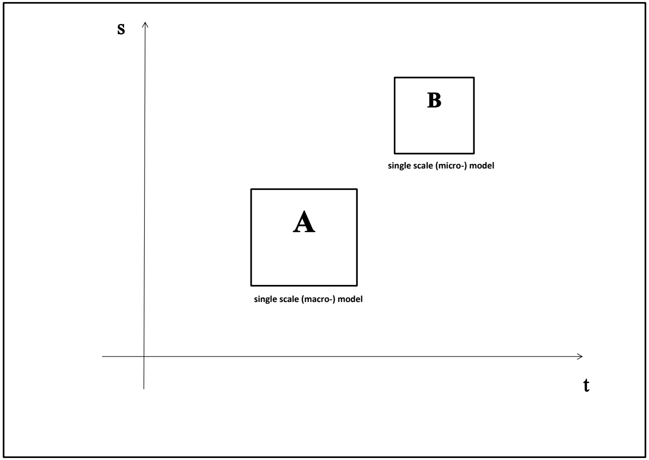


Figure : Creation of a scale separation map

## MUSCLE: Illustration of the Modeling Methodology (2/4)

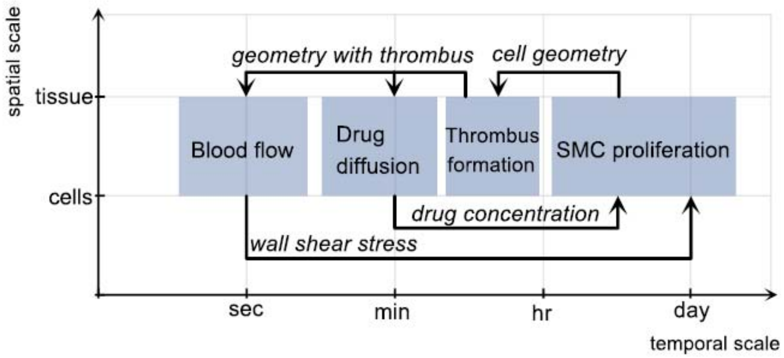


Figure : Scale separation map of an In-stent Restenosis: exhibits only time scale separation (no spatial scale separation: all submodels act at the same spatial scale) Source: J. Borgdorff et al., Foundations of distributed multiscale computing, J. Parallel Distrib. Comput., (2013)

## MUSCLE: Illustration of the Modeling Methodology (3/4)

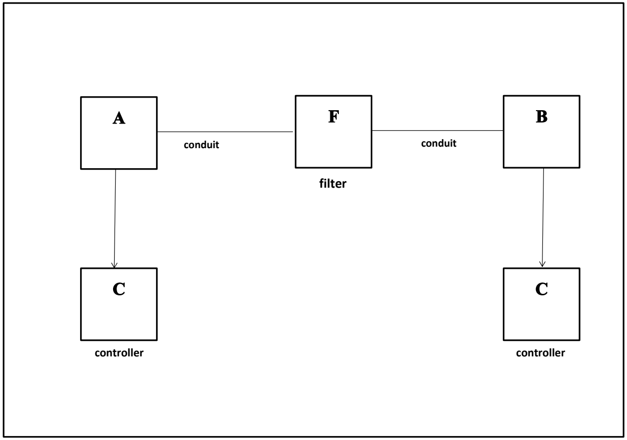


Figure : Implementation of the submodels using controllers (APIs) and creation of a multiscale model - including filters, mappers, etc. for scale bridging, data conversions, etc.

# MUSCLE: Illustration of the Modeling Methodology (4/4)

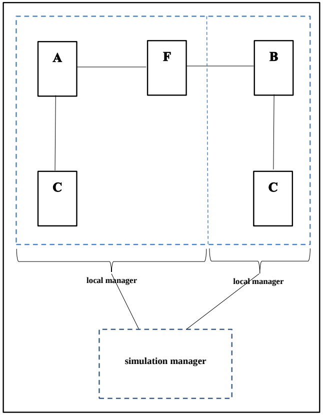


Figure : Mapping of the Application onto the Simulation Engine (Local and Simulation Managers)

## UML/MARTE: Illustration of the Modeling Methodology (the Requirement Phase)

- (system) models are subdivided into three sub-models (Y structure)
  1. Platform Independent Model (PIM), intended to represent various views of the system (data, functional, application, concurrency, communication, memory space)
  1. Platform Description Model (PDM), intended to model the execution platform supporting the PIM
  1. Platform Specific Model (PSM), intended to model the allocation of PIM to PDM
- PIM and PDM developers use Hardware/Software Resource Models (HRM / **SRM**) sub-profiles acting as an API

## UML/MARTE: More on the SRM sub-profile

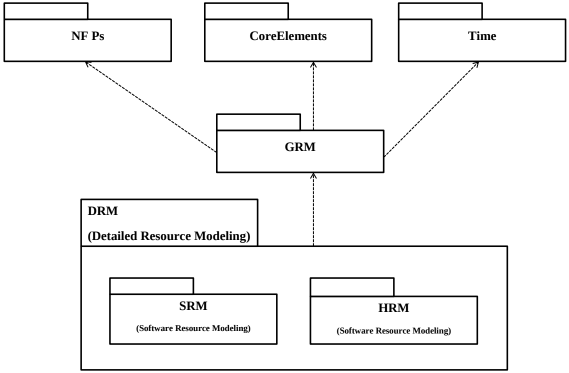

NF Ps
CoreElements
Time
GRM
DRM
(Detailed Resource Modeling)
SRM
(Software Resource Modeling)
HRM
(Software Resource Modeling)

Figure : Source: OMG Document Number: ptc/2008-06-09

## Bridging the Gap: Creation of a Specific SRM for DMMS

- the analysis of domain model of the SRM and of the domain model of the resource modeling elements used by the DMMS show some similarities at the level of the used resource types
- this led to the idea of building a specific SRM for the DMMS by merely extending the SRM (and not starting from the scratch)

## Extending the SRM

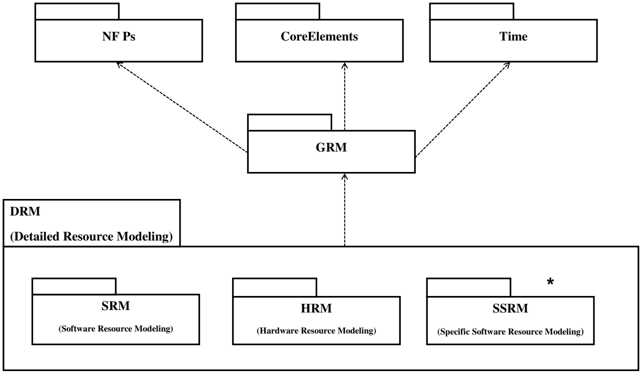

NF Ps
CoreElements
Time
GRM
DRM
(Detailed Resource Modeling)
SRM
(Software Resource Modeling)
HRM
(Hardware Resource Modeling)
*
SSRM
(Specific Software Resource Modeling)

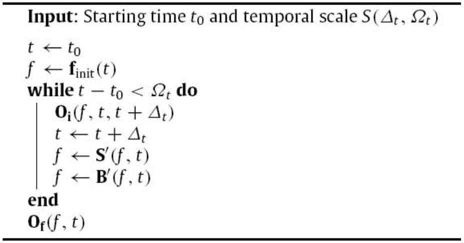
Figure : The SSRM

## The SSRM Resource Types

- we need to
    - specify more precisely the domain model (of this SSRM) in order to
    - "extract" (from this model) potential stereotypes
- this necessitates to abstract the DMMS core elements (controllers, filters, mappers, conduits, etc.)
- here we consider
    - controllers, and
    - conduits
- a controller (in MUSCLE) is implemented by a so-called Submodel Execution Loop (SEL)

## (Time-Driven) Submodel Execution Loop (SEL)

- $f$: current state of the submodel
- Operators used in the SEL
  - $f_{init}$: for initialization of the state
  - $S'$: for solving one modeled step
  - $O_i$ and $O_f$: for an observation of an intermediate and final state



```
Input: Starting time $t_0$ and temporal scale $S(\Delta_t, \Omega_t)$
$t \leftarrow t_0$
$f \leftarrow \mathbf{f}_{\text{init}}(t)$
while $t - t_0 < \Omega_t$ do
    $\mathbf{O}_{\mathbf{i}}(f, t, t + \Delta_t)$
    $t \leftarrow t + \Delta_t$
    $f \leftarrow \mathbf{S}'(f, t)$
    $f \leftarrow \mathbf{B}'(f, t)$
end
$\mathbf{O}_{\mathbf{f}}(f, t)$
```

Figure : Source: J. Borgdorff et al., Foundations of Distributed Multiscale Computing, Formalization, Specification, and Analysis, J. Parallel Distrib. Comput. 73(2013)

## Submodel Execution Loop (an Abstraction)

```
$f_{init}$
 while {
    $O_i$ ;
    $S$
 }
$O_f$
}
```

Figure : SEL abstracted to essential operators

## Recall: (Part of) the Domain Model of SRM

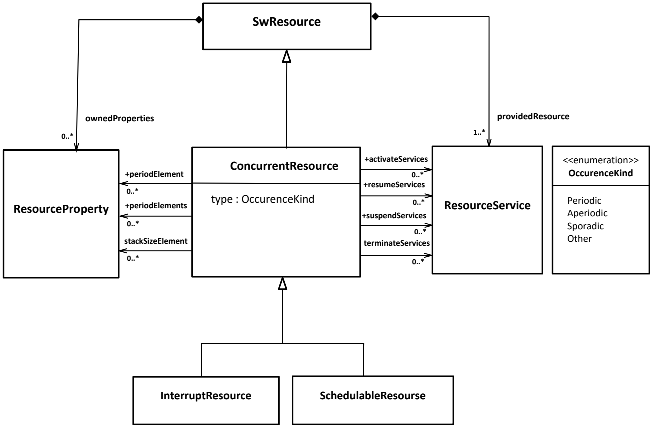


Figure : The Schedulable Concurrent Resource

## Extending the SRM Domain Model (1/2)

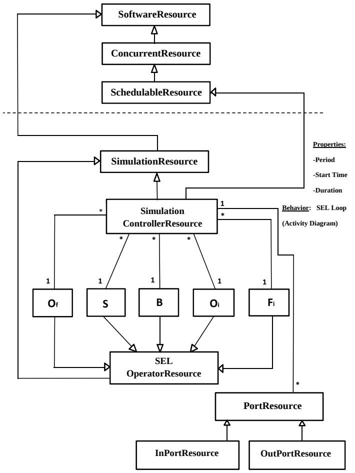


Figure : The Controller as a Schedulable Concurrent Resource

# Extending the SRM Domain Model (2/2)

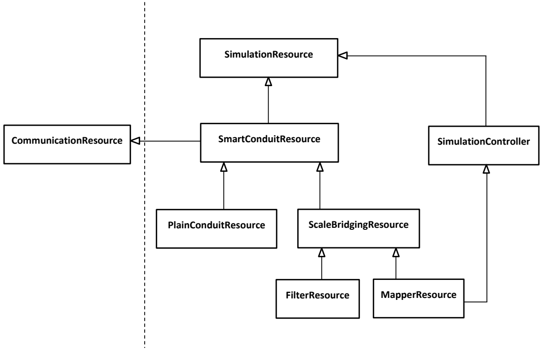


Figure : The Conduit as a Simulation Communication Resource, Mapper is also a Controller

# Using SSRM in DMMS

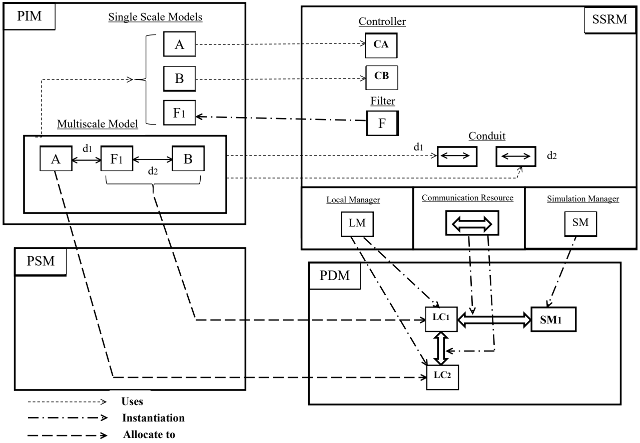


Figure : Illustration using MUSCLE

## CSP-OZ (the Design Phase)

- CSP-OZ (an extension of Object-Z with the notion of communication channels and CSP syntax) with a failure-divergence semantics
  - property analysis (DMMS community claim to face, among other verification issues, the crucial problem of deadlock)
- the deliverables produced at the requirement level are models that describe the targeted application at a high level of abstraction
- those produced by UML/MARTE form a "collection" of components inteconnected via UML/MARTE connectors
- at the lowest level of such diagrams we find RtUnits and PpUnits
  - RtUnits are active classes characterized by the services they provide, their communication ports, their attributes and also by their behavior usually expressed in terms of protocol state machines
  - PpUnits (protected passive units) are similar to RtUnits except that they do not own an internal behavior (passive classes)

## From UML/MARTE to CSP-OZ

- RtUnits and PpUnits are transformed into their corresponding CSP-OZ classes
  - an RtUnit service is mapped to a CSP-OZ communication schema
  - all RtUnit/PpUnit attributes are mapped to a CSP-OZ data schema
  - an RtUnit/PpUnit port is mapped to a CSP-OZ channel
  - a protocol state machine associated with an RtUnit is mapped to a CSP process (i.e., the CSP part of the corresponding CSP-OZ class)
- a flat component diagram (PpUnits/RtUnits interconnected with MARTE connectors through ports) is transformed into a CSP-OZ system class
  - a set of objects that belong to the resulting CSP-OZ classes
  - a set of CSP channels
  - a CSP process describing the overall behaviour of the CSP-OZ system class

## PyCSP (the Implementation Phase)

- PyCSP is a CSP library for Python
  - supports the core abstractions of CSP (i.e., process and channel)
  - the transformation of a CSP-OZ into PyCSP code is straightforward
- CSP-OZ class transformation
  - a channel (defined in the CSP-OZ class) is mapped to a PyCSP channel
  - a communication schema (defined in a CSP-OZ class) is mapped to a Python function
  - the process part (of a CSP-OZ class) is mapped to (an individual) PyCSP process
- a (CSP-OZ) system class is mapped to a PyCSP process
  - (individual) PyCSP processes (resulting from corresponding CSP-OZ classes) are combined using relevant CSP operators

## Sematic Issues

- institution theory
- some of the adopted formalisms are backed by institutions (others are in progress)
- we adopt a similar approach as the one presented in the position paper: Towards an Institutional Framework for Heterogeneous Formal Development in UML (Knapp, Mossakowski, Roggenbach, 2015)

## Sematic Issues (Continued)

- types only are considered

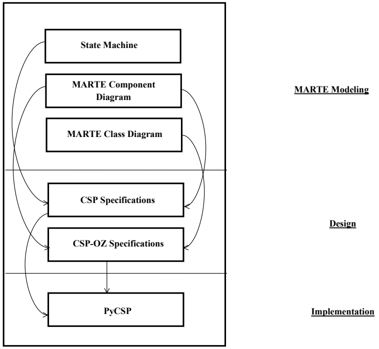


Figure : Institution comorphisms between MARTE diagrams and languages

## Conclusions and Future Work

- extending MARTE with a new subprofile (SSRM) dedicated to DMMS (intended to define specific modeling resources that capture DMMS core concepts)
- setting a formal semantic framework for the proposed development methodology (ensuring a sound integration of UML/MARTE, CSP-OZ and PyCSP)
- state machines, CSP, OZ do have institutions
- remains to build institutions for
    - PyCSP (from scratch)
    - UML/MARTE profile: build on what is planned in the work by (Knapp, Mossakowski, Roggenbach, 2015)